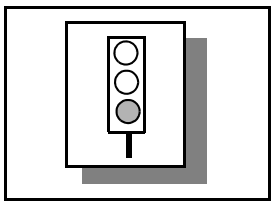
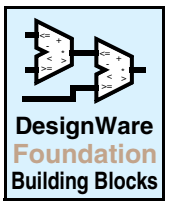




DW_arb_rr

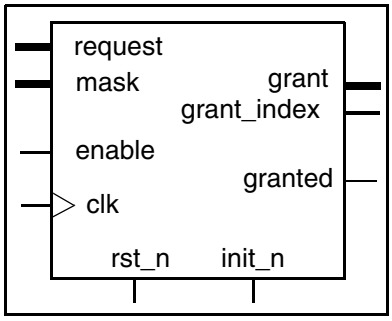
Arbiter with Round Robin Priority Scheme

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

- Parameter controlled number of clients
- Programmable `mask` for all clients
- Parameter controlled optional registered output
- Parameter controlled `grant_index` coding scheme
- Single cycle arbitration

Revision History



Applications

- Control application
- Networking
- Bus interfaces

Description

DW_arb_rr implements a round-robin architecture with a parameterized number of clients. For each client, the lowest number client has priority at contention, and at each subsequent event the next higher numbered client has the highest priority until the highest client has been service, and then starts over.

By setting the desired bit of the `mask` input, the input corresponding to the set bit of `mask` will be blocked, that is, no consideration for arbitration occurs for that client.

All inputs are synchronized with `clk`.

The arbiter provides the status flag `granted`, showing when the resource is in use, and *n*-bit `grant` indicating which client has been granted the resource, and also `grant_index`, the binary value of `grant`. The `grant_index` behavior depends on the value of the *index_mode* parameter.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Active-low asynchronous reset
init_n	1 bit	Input	Synchronous reset for all registers (active low)

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
enable	1 bit	Input	Controls when arbitration operates: ■ 1: Enables arbitration ■ 0: Disables arbitration, releases arbitration client, and puts DW_arb_rr in an idle state
request	*n* bits	Input	Input request from clients
mask	*n* bits	Input	Active high input to mask specific clients By setting `mask`(*i*) = 1, `request`(*i*) is masked.
granted	1 bit	Output	Flag to indicate that arbiter has issued a grant to one of the clients
grant	*n* bits	Output	Grant output, one bit per client
grant_index	ceil($\log_2$(*n* + mod2(*index_mode*)) bits	Output	Index of the client that has been currently granted

Table 1-2 Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
output_mode	0 or 1 Default: 1	■ 1: Includes registers at the outputs ■ 0: Contains no output registers
index_mode[a]	0 to 2 Default: 0	■ 0 or 1: Causes `grant_index` value to be `grant` bit position plus 1 ■ 2: Causes `grant_index` values to be the bit position of `grant`

a. The *index_mode* parameter does not exist in DW_arb_rr prior to the 201206.3 DWBB release. For backward compatibility, the default *index_mode* = 0 behavior is the same as DW_arb_rr prior to the 201206.3 DWBB release.

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis Model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW05.DW_ARB_rr_SIM_CFG	Design unit name for VHDL simulation

Table 1-4 Simulation Models (Continued)

Model	Function
dw/dw05/DW_arb_rr_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_arb_rr.v	Verilog simulation model source code

Functional Description

DW_arb_rr can have from 2 to 32 requesting clients. Any client can be masked by setting the corresponding bit in the mask input. For each client input, the lowest numbered client initially has priority. At contention, when two or more clients simultaneously request the resource, the arbiter initially grants to the lowest numbered client. Once that client releases its request, the arbiter sets the priority to the next sequentially numbered client. If the next request is also multiple simultaneous requests, the next highest client has priority and the priority is passed 'round robin' to each client in turn.

Arbitration operates when the `enable` input is high and it is disabled when the `enable` input is low. Setting the enable input low drops that client and the arbiter goes into an idle state.

Index Mode Specification

The value of the *index_mode* parameter controls the operation of the `grant_index` output. When *index_mode* is set to either 0 or 1, the `grant_index` output will use a value that is one greater than the bit index of the granted client when a `grant` is active and the value 0 (all zeros) when no `grant` is active. When *index_mode* is set to 2, the `grant_index` output will use the bit index of the granted client when a `grant` is active and 0 (all zeros) when no `grant` is active.

The difference between *index_mode* 0 and *index_mode* 1 is in the sizing of the `grant_index` output port. For *index_mode* = 0, the `grant_index` port is ceil(log2(*n*)) bits wide, but for *index_mode* = 1 `grant_index` is ceil(log2(*n*+1)) bits wide. The difference only appears when the number of clients (as specified by the value of parameter *n*) is an integer power of 2 (such as 2, 4, 8, 16, 32) when operation with *index_mode* = 0 does not have the extra bit of width required to identify the last client while operation with *index_mode* = 1 will have enough bits to indicate all clients uniquely (see Table 1-5 and Table 1-6).

Table 1-5 How *index_mode* Affects the `grant_index` Output Shown with Parameter *n = 8*

`grant` output	`grant_index` output with *index_mode* = 0 (size = ceil(log2(8)) = 3)	`grant_index` output with *index_mode* = 1 (size = ceil(log2(8+1)) = 4)	`grant_index` output with *index_mode* = 2 (size = ceil(log2(8)) = 3)
00000001	001	0001	000
00000010	010	0010	001
00000100	011	0011	010
00001000	100	0100	011
00010000	101	0101	100
00100000	110	0110	101
01000000	111	0111	110
10000000	000	1000	111

Table 1-5 How *index_mode* Affects the `grant_index` Output Shown with Parameter *n = 8* (Continued)

`grant` output	`grant_index` output with *index_mode* = 0 (size = ceil(log2(8)) = 3)	`grant_index` output with *index_mode* = 1 (size = ceil(log2(8+1)) = 4)	`grant_index` output with *index_mode* = 2 (size = ceil(log2(8)) = 3)
00000000	000	0000	000

Table 1-6 How *index_mode* Affects the `grant_index` Output Shown with Parameter *n = 7*

`grant` output	`grant_index` output with *index_mode* = 0 (size = ceil(log2(7)) = 3)	`grant_index` output with *index_mode* = 1 (size = ceil(log2(7+1)) = 3)	`grant_index` output with *index_mode* = 2 (size = ceil(log2(7)) = 3)
0000001	001	001	000
0000010	010	010	001
0000100	011	011	010
0001000	100	100	011
0010000	101	101	100
0100000	110	110	101
1000000	111	111	110
0000000	000	000	000

Arbiter Status

All the input requests from the arbiter clients are assumed to be synchronous to the arbiter clock signal clk. Table 1-7 shows a detailed description of the granted flag.

Table 1-7 Arbiter Status Flag

Flag	Characteristic	Description
granted	If `granted` is active, there is at least one active request at the input of the arbiter.	The `granted` output, active high, indicates that the grant of resources is to one of the actively requesting inputs.

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

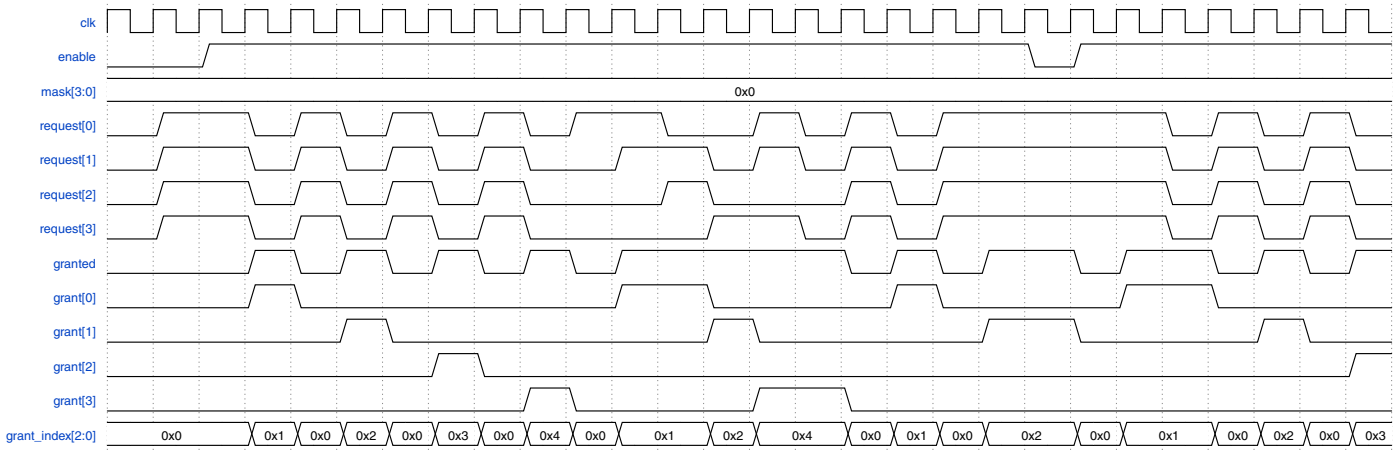
      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

Timing Waveforms

Figure 1-1 shows waveforms when DW_arb_rr is operating with *output_mode* = 1 and *index_mode* = 1 (assuming all `mask` bits are '0'):

Figure 1-1 Waveform of dw_arb_rr



Related Topics

- Application Specific – Control Logic Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.DW_Foundation_comp.all;

entity DW_arb_rr_inst is
      generic (
        inst_n : NATURAL := 4;
        inst_output_mode : NATURAL := 1;
        inst_index_mode : NATURAL := 0
        );
      port (
        inst_clk : in std_logic;
        inst_rst_n : in std_logic;
        inst_init_n : in std_logic;
        inst_enable : in std_logic;
        inst_request : in std_logic_vector(inst_n-1 downto 0);
        inst_mask : in std_logic_vector(inst_n-1 downto 0);
        granted_inst : out std_logic;
        grant_inst : out std_logic_vector(inst_n-1 downto 0);
       grant_index_inst : out std_logic_vector(bit_width(inst_n + (inst_index_mode mod
2))-1 downto 0)
        );
    end DW_arb_rr_inst;


architecture inst of DW_arb_rr_inst is

begin

    -- Instance of DW_arb_rr
    U1 : DW_arb_rr
    generic map ( n => inst_n,
                output_mode => inst_output_mode,
                index_mode => inst_index_mode )
    port map ( clk => inst_clk,
                rst_n => inst_rst_n,
                init_n => inst_init_n,
                enable => inst_enable,
                request => inst_request,
                mask => inst_mask,
                granted => granted_inst,
                grant => grant_inst,
                grant_index => grant_index_inst );


end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_arb_rr_inst (
                inst_clk,
                inst_rst_n,
                inst_init_n,
                inst_enable,
                inst_request,

          inst_mask,
                granted_inst,
                grant_inst,
                grant_index_inst );

parameter n = 4;
parameter output_mode = 1;
parameter index_mode = 0;

`define   bit_width_n 2  // ceil(log2(n + (index_mode % 2)))

input inst_clk;
input inst_rst_n;
input inst_init_n;
input inst_enable;
input [n-1 : 0] inst_request;
input [n-1 : 0] inst_mask;
output granted_inst;
output [n-1 : 0] grant_inst;
output [`bit_width_n-1 : 0] grant_index_inst;

    // Instance of DW_arb_rr
    DW_arb_rr #(n,
                output_mode,
                index_mode)
      U1 (  .clk(inst_clk),
                .rst_n(inst_rst_n),
                .init_n(inst_init_n),
                .enable(inst_enable),
                .request(inst_request),
                .mask(inst_mask),
                .granted(granted_inst),
                .grant(grant_inst),
                .grant_index(grant_index_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2023	DWBB_202212.5	■ Corrected the description of the enable input in Table 1-2 on page 2 and in "Functional Description" on page 3 ■ Updated waveforms in Figure 1-1 on page 6
July 2020	DWBB_201912.5	■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 5 and added the DW_SUPPRESS_WARN macro
October 2019	DWBB_201903.5	■ Added the "Disabling Clock Monitor Messages" section
March 2019	DWBB_201903.0	■ Removed minPower designation from this datasheet ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com